REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Cited Art

The Action cites The Object Management Group, entitled "The Common Object Request Broker: Architecture and Specification CORBA," Revision 2.0, July 1995 ("CORBA").

The Action also cites Steinman, J., entitled "Incremental State Saving in Speedes Using C++," Proceedings of the 1993 Winter Simulation Conference ("Steinman").

Further, the Action cites Bruce, D., "The Treatment of State in Optimistic Systems," IEEE, pp. 40-49 ("Bruce").

Finally, the Action cites United States Patent No. 5,889,957, to Ratner et al. ("Ratner"), entitled "Method and Apparatus for Context Sensitive Pathsend," and United States Patent No. 5,765,174, to Bishop, entitled "System and Method for Distributed Object Resource Management" ("Bishop").

Amended Claims

According to the Examiner's recommendations, Applicants have amended claim 8 to include "all the limitations of" claim 5. *See* Office Action, page 5, ¶ 6. This places claim 8 in condition for allowance. Further, claims 9 and 10 are allowable since they depend from claim 8. Such action is respectfully requested.

Further, as suggested by the Examiner, Applicants have amended claim 15 to include all the limitations of claim 13. This places claim 15 in condition for allowance. Further, claims 16 and 17 are allowable since they depend from claim 15. Such action is respectfully requested.

Patentability of Claims 1-7, 11-14, and 18-21 over CORBA, Steinman, Bruce, Ratner and Bishop, under § 103

The Office Action rejects claims 1, 3-7, 11-14, and 18-21 under 35 U.S.C. § 103(a) as unpatentable over CORBA in view of Ratner and Steinman. The Action rejects claim 2 under 35 U.S.C. § 103(a) as unpatentable over CORBA in view of Ratner, Steinman, and Bishop. Applicants respectfully traverse the rejections.

Claim 1

Claim 1 is directed to a method of enhancing server scalability comprising destroying the state of the application component in response to the indication from the application component. Specifically, claim 1 recites,

> 1. (Thrice Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:
>
> executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;
>
> providing an interface for the operating service to receive an indication from the application component that the work is complete;
>
> maintaining the state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and
>
> destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client.
>
> (Emphasis added.)

Applicants respectfully submit that the Examiner has failed to meet the burden of establishing obviousness because the cited art lacks any teaching that would suggest "destroying the state of the application component in response to the indication from the application component."

The Office admits that CORBA fails to teach "(3) destroying is in response to an indication without action by client." *See* Office Action, page 3, ¶ 4. This is true, however, the statement omits which object is being destroyed and which object is requesting its destruction. Subsequently, when the Office asserts Ratner in the rejection of claim 1, this feature is not addressed with specificity. Specifically, the Office fails to cite any reference that teaches or suggests "destroying the state of the application component in response to the indication from the application component ... without action by the client."

It is antithetical to object oriented design for (1) an object to request its own destruction, (2) for an object to request its own destruction while clients hold a reference, or (3) for objects to request their own destruction while they have a positive reference count. Rather, an object is

garbage collected by a run-time system after a client has released a reference to the object. For example, Bishop discloses that objects should not be destroyed without the permission of a client holding a reference to the object. *See* Bishop, col. 4, lines 16-22; col. 1, lines 33-35 and lines 49-55. Thus, the prior art teaches away from "destroying the state of the application component in response to the indication from the application component."

The Office asserts that "destroying the state of the application component in response to the indication from the application component ... without action by the client" is disclosed in Ratner. Applicants respectfully disagree. Applicants are unable to find any statement in Ratner that teaches or suggests an object calling an interface to destroy its own state and "without action by the client." Ratner discloses that an "originating process" can create a dialogue, and can abort a dialogue. *See* col. 7, line 1; col. 7, line 43. Further, a "receiving process" can break a dialogue with a return code of FeOK. *See* col. 7, line 30. However, nowhere does Ratner teach or suggest that a dialogue can request destruction of its own state. Further, a dialogue is not an application component. Rather, a dialogue is "a logical connection between the server and client." *See* Ratner, Abstract. The Office has failed to cite any reference that teaches or suggests "destroying the state of the application component in response to the indication from the application component ... without action by the client."

It is not unusual in legacy communications protocols to provide a way for an originating process or a receiving process to end a communications link. Although Ratner discusses a way for either the originating process or receiving process to end a logical connection (dialog), that is not what claim 1 recites. Rather, claim 1 recites "destroying the state of the application component in response to the indication from the application component ... without action by the client." In the object-oriented paradigm it is antithetical to either (1) destroy the state of an "application component in response to the indication from the application component," or (2) to destroy an "application component," "without action by the client," and "in response to the indication from the application component." The art cited by the Office fails to teach or suggest either of these two features.

Ratner discusses killing a logical connection (dialog) between an originating and a receiving process. The dialogue supports communication between the processes, it does not contain the state of the processes. The connection may contain information used to direct messages between processes, but the connection (dialog) discussed in Ratner does not contain

the "state of the application component," and the dialog is not calling for the destruction of its own state. *See* Ratner, col. 10, lines 16-29. A skilled artisan reading about an originating or receiving process killing a logical connection could not reasonably be expected to learn "destroying the state of the application component in response to the indication from the application component ... without action by the client."

In the discussion of claim 21, the Office asserts that Ratner discloses that the "server triggers the breaking of dialog and destruction/deallocation of server state (information maintained in the Dialogue Control Block; and a session/dialog is broken immediately after the server sends out a message other than the special error code (which is the code for "continue this session")." *See* Office Action, page 3, lines 1-4.

Thus, the Office's position equates a "Dialogue Control Block" with "the state of the application component." Applicants respectfully disagree. As understood by Applicants, the Dialog Control Block (DCB) discussed in Ratner, is used to identify and direct context sensitive messages from a client to a server. *See* Ratner, col. 10, lines 43-47. However, in Ratner, the DCB is not disclosed as maintaining a server component's state. Applicants respectfully submit that the state of the application component is not destroyed when a message channel is destroyed.

Further, Ratner, at most, is disclosing that an originating or receiving process can cause destruction of a DCB. However, nowhere does Ratner teach or suggest "destroying" an application component in response to its own indication. In Ratner, the DCB is a data structure that is destroyed based on a request of a process. Destruction of a DCB data structure based on the request of a process fails to teach or suggest destruction of an application component based on the request of an application component.

Finally, Steinman fails to teach or suggest the recited feature.

In order to sustain the burden of obviousness, the prior art reference (or references when combined) must teach or suggest *all the claim limitations*. MPEP §2142. Since the art cited by the Office either alone or in combination fails to teach or suggest "destroying the state of the application component in response to the indication from the application component ... without action by the client," the Office has failed to establish obviousness.

For at least this reason, claim 1 is allowable. Such action is respectfully requested.

Dependent Claims 3 and 4

Dependent claims 3 and 4 recite additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claims 3 and 4 are not belabored at this point. For at least the reasons stated in claim 1, claim 3 and 4 should be allowed. Such action is respectfully requested.

Claim 2

Claim 2 is allowable for all the reasons stated above in claim 1. However, Applicant's separately addresses claim 2 in order to discuss Bishop.

The Office Action asserts that Bishop teaches only the last phrase from claim 2. However, Applicant addresses Bishop more broadly because Bishop helps establish how the prior art teaches away from the recited features.

In Bishop the client *allows* the operating service to garbage collect an object by calling the make weak function. *See* Bishop, col. 1, lines 33-35, and lines 49-55. Therefore, the client must take *action*, and the object being destroyed by garbage collection takes *no action*. Thus, Bishop teaches away from "destroying the state of the application component in response to the indication from the application component ... without action by the client." In Bishop, the operating service garbage collects objects without any indication from the object itself, and only after the client calls the make weak function. Nowhere does Bishop teach or suggest that an object can request destruction of itself while a "client retains a ... reference."

Bishop exemplifies how the prior art views object destruction. It is contrary to the object oriented canon for (1) an object to request its own destruction, (2) for an object to request its own destruction while clients hold a reference, and (3) for objects to be destroyed at their own request while they have a positive reference count. Rather, objects are garbage collected by a run-time system after clients have released or reduced their holding power on an object. For example, Bishop discloses that objects should not be destroyed while clients maintain references to the object. *See* Bishop, col. 4, lines 16-22. Bishop discusses allowing garbage collection of an object if a client of the object calls a make weak function. Thus, the prior art teaches away from "destroying the state of the application component in response to the indication from the application component ... without action by the client ... while the client retains a client's reference."

For at least this reason, claim 2 should be allowed in its present form.

Claim 5

Claim 5 is directed to a computer operating environment for scalable, component-based server applications, comprising "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state." Specifically, claim 5 recites,

> 5. (Unchanged) In a computer, a computer operating environment for scalable, component-based server applications, comprising:
>
> a run-time service for executing an application component in a process, the application component having a state and implementing a set of functions;
>
> an instance creation service operative, responsive to a request of a client, to return a reference to the application component through the run-time service to the client, whereby the client calls functions of the application component indirectly through the run-time service using the reference to initiate work by the application component; and
>
> the run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client. (Emphasis added.)

Applicants respectfully submit that the Examiner has not met the burden of establishing obviousness of the claimed method, because the cited art lacks any teaching that would suggest "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state."

In asserting a rejection against claim 5, the Office directs Applicant to the Office's discussion of claim 1. In the discussion of claim 1, the Office admits that CORBA fails to teach "(3) destroying is in response to an indication without action by client." *See* Office Action, page 3, ¶ 4. This is true; however, the statement omits what object is being destroyed and which object is requesting its destruction. Subsequently, when the Office asserts Ratner in the rejection of claim 1, this feature is not addressed with specificity. Specifically, the Office fails to cite any reference that teaches or suggests "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state."

It is contrary to the object oriented canon for (1) an object to request its own destruction, (2) for an object to request its own destruction while clients hold a reference, or (3) for objects to request their own destruction while they have a positive reference count. Rather, an object is garbage collected by a run-time system after a client has released a reference to the object. For example, Bishop discloses that objects should not be destroyed without the permission of a client holding a reference to the object. *See* Bishop, col. 4, lines 16-22; col. 1, lines 33-35 and lines 49-55. Thus, the prior art teaches away from "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state."

The Office asserts that this feature is disclosed in Ratner. Applicants respectfully disagree. Applicants are unable to find any statement in Ratner that teaches or suggests "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state." Ratner discloses that an "originating process" can create a dialogue, and can abort a dialogue. *See* col. 7, line 1; col. 7, line 43. Further, a "receiving process" can break a dialogue with a return code of FeOK. *See* col. 7, line 30. However, nowhere does Ratner teach or suggest that a dialogue can request destruction of its own state. Further, a dialogue is not an application component. Rather, a dialogue is "a logical connection between the server and client." *See* Ratner, Abstract.

It is not unusual in legacy communications protocols to provide a way for an originating process or a receiving process to end a communications link. However, this is not what the claim recites. In the object-oriented paradigm it is antithetical for an object to request its own destruction.

Ratner discusses killing a logical connection between an originating and a receiving process. The logical connection supports communication between the two processes. Ratner fails to teach or suggest "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state."

Further, a dialogue is not an application component. Rather, a dialogue is "a logical connection between the server and client." *See* Ratner, Abstract. In Ratner, the originating or receiving process can "kill" the connection; but Ratner fails to teach or suggest "responsive to an indication from the application component that the application component has completed the

work for the client, to destroy the application component's state." A skilled artisan reading about an originating or receiving process killing a logical connection could not reasonably be expected to evince "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state."

In the discussion of claim 21, the Office asserts that Ratner discloses that the "server triggers the breaking of dialog and destruction/deallocation of server state (information maintained in the Dialogue Control Block; and a session/dialog is broken immediately after the server sends out a message other than the special error code (which is the code for "continue this session")." *See* Office Action, page 3, lines 1-4.

Thus, the Office's position equates a "Dialogue Control Block" with "the state of the application component." Applicants respectfully disagree. As understood by Applicants, the Dialog Control Block (DCB) discussed in Ratner, is used to identify and direct context sensitive messages from a client to a server. *See* Ratner, col. 10, lines 43-47. However, in Ratner, the DCB is not disclosed as maintaining a server component's state. Applicants respectfully submit that the state of the application component is not destroyed when a message channel is destroyed.

Further, Ratner, at most, is disclosing that an originating or receiving process can cause destruction of a DCB. However, nowhere does Ratner teach or suggest "destroying" an application component in response to its own indication. In Ratner, the DCB is a data structure that is destroyed based on a request of a process. Destruction of a DCB data structure based on actions of a process fails to teach or suggest destruction of an application component based on the actions of an application component.

In order to sustain the burden of obviousness, the prior art reference (or references when combined) must teach or suggest *all the claim limitations*. MPEP §2142. Since the art cited by the Office either alone or in combination fails to teach or suggest "responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state," the Office has failed to establish obviousness.

For at least this reason, claim 5 is allowable. Such action is respectfully requested.

Dependent Claims 6, 7, 11, and 12

Dependent claims 6, 7, 11, and 12 recite additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate

patentability of dependent claims 6, 7, 11, and 12 are not belabored at this point. For at least the reasons stated in claim 5, claims 6, 7, 11, and 12 should be allowed. Such action is respectfully requested.

Claim 13

Claim 13 is directed to a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising discarding the processing state of the component responsive to the component indicating completion of the work. Specifically, claim 13 recites,

> 13. (Once Amended) In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:
> encapsulating function code and a processing state for the work in a component;
> providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
> receiving an indication from the component that the work by the component is complete; and
> discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.
> (Emphasis added.)

Applicants respectfully submit that the Examiner has not met the burden of establishing obviousness of the claimed method, because the cited art lacks any teaching that would suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client."

In asserting a rejection against claim 13, the Office directs Applicant to the Office's discussion of claim 1. In the discussion of claim 1, the Office admits that CORBA fails to teach "(3) destroying is in response to an indication without action by client." *See* Office Action, page 3, ¶ 4. This is true; however, the statement omits which object is being destroyed and which object is requesting its destruction. Subsequently, when the Office discusses Ratner, this feature is not addressed with specificity. Specifically, the Office fails to cite any reference that teaches or suggests "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client."

It is contrary to the object oriented canon for (1) an object to request its own destruction, (2) for an object to request its own destruction while clients hold a reference, and (3) for objects to request their own destruction while they have a positive reference count. Rather, an object is garbage collected by a run-time system after a client has released a reference to the object. For example, Bishop discloses that objects should not be destroyed without permission of a client holding a reference to the object. *See* Bishop, col. 4, lines 16-22; col. 1, lines 33-35 and lines 49-55. Thus, the prior art teaches away from "discarding the processing state of the component responsive to the component indicating completion of the work."

The Office asserts that this limitation is disclosed in Ratner. Applicants respectfully disagree. Applicants are unable to find any statement in Ratner that teaches or suggests a discarding the processing state of a based on an indication of the component. Ratner discloses that an "originating process" can create a dialogue, and can abort a dialogue. *See* col. 7, line 1; col. 7, line 43. Further, a "receiving process" can break a dialogue with a return code of FeOK. *See* col. 7, line 30. However, nowhere does Ratner teach or suggest that a dialogue can request destruction of its own state.

Further, a dialogue is not an application component. Rather, a dialogue is "a logical connection between the server and client." *See* Ratner, Abstract. In Ratner, the originating or receiving process can "kill" the connection; but Ratner fails to teach or suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client." A skilled artisan reading about an originating or receiving process killing a logical connection could not reasonably be expected to evince "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client."

In order to sustain the burden of obviousness, the prior art reference (or references when combined) must teach or suggest *all the claim limitations*. MPEP §2142. Since the art cited by the Office either alone or in combination fails to teach or suggest "discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client," the Office has failed to establish obviousness.

For at least this reason, claim 13 is allowable. Such action is respectfully requested.

Dependent Claim 14

Dependent claim 14 recites additional patentably distinct subject matter. However, in view of the foregoing discussion and amendments, the merits of the separate patentability of dependent claims 14 is not belabored at this point. For at least the reasons stated in claim 13, claim 14 should be allowed. Such action is respectfully requested.

Claim 18

Claim 18 is directed to a system service for providing an execution environment for scalable application components, comprising code for destroying the processing state of the application program in response to the indication from the application component without action from the client program. Specifically, claim 18 recites,

> 18. (Unchanged) In a computer, a system service for providing an execution environment for scalable application components, comprising:
>
> code responsive to a request from a client program to create an application component for returning to the client program a reference through the system service to the application component;
>
> code responsive to a call from the client program using the reference for initiating processing of work by the application component, the application component producing a processing state during processing the work;
>
> code for receiving an indication from the application component that processing by the application component of the work is complete; and
>
> code for destroying the processing state of the application program in response to the indication from the application component without action from the client program.
>
> (Emphasis added.)

Applicants respectfully submit that the Examiner has not met the burden of establishing obviousness of the claimed method, because the cited art lacks any teaching that would suggest "code for destroying the processing state of the application program in response to the indication from the application component without action from the client program."

The Office directs Applicants to claim 5 for the asserted rejection of claim 18. For at least the reasons stated in claim 5, the art cited by the Office fails to teach or suggest "destroying the processing state of the application program in response to the indication from the application component without action from the client program."

For at least this reason, claim 18 is allowable. Such action is respectfully requested.

Dependent Claims 19 and 20

Dependent claims 19 and 20 recite additional patentably distinct subject matter. However, in view of the foregoing discussion, the merits of the separate patentability of dependent claims 19 and 20 are not belabored at this point. For at least the reasons stated in claim 18, claims 19 and 20 should be allowed. Such action is respectfully requested.

Claim 21

Claim 21 is directed to a method of enhancing scalability of server applications comprising "destroying the state of the application component in response to the indication from the application component without action by the client." Specifically, claim 21 recites,

> 21. (Once Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:
>
> executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;
>
> maintaining the state in the main memory between the method invocations of the function code by the client in the absence of an indication from the application component that the work is complete; and
>
> destroying the state by the operating service in response to an indication from the application component without action by the client, such that the destroyed state is not persistent.
>
> (Emphasis added.)

Applicants respectfully submit that the Office has failed to meet the burden of establishing obviousness because the cited art lacks any teaching that would suggest "destroying the state of the application component in response to the indication from the application component without action by the client."

The Office asserts that Ratner discloses that the "server triggers the breaking of dialog and destruction/deallocation of server state (information maintained in the Dialogue Control Block; and a session/dialog is broken immediately after the server sends out a message other than the special error code (which is the code for "continue this session")." *See* Office Action, page 3, lines 1-4.

Thus, the Office's position equates a "Dialogue Control Block" with "the state of the application component." Applicants respectfully disagree. As understood by Applicants, the

Dialog Control Block (DCB) discussed in Ratner, is used to identify and direct context sensitive messages from a client to a server. *See* Ratner, col. 10, lines 43-47. However, in Ratner, the DCB is not disclosed as maintaining the server's state. Applicants respectfully submit that the state of the application component is not necessarily destroyed when a message channel is destroyed.

Further, Ratner at most, is disclosing that an originating or receiving process can cause destruction of a DCB. However, nowhere does Ratner teach or suggest "destroying" an application component in response to its own indication. In Ratner, the DCB is destroyed based on the actions of another. Destruction of a DCB based on actions of a process fails to teach or suggest destruction of an application component based on the actions of an application component.

In order to sustain the burden of obviousness, the prior art reference (or references when combined) must teach or suggest *all the claim limitations*. MPEP §2142. Since the art cited by the Office either alone or in combination fails to teach or suggest "destroying the state of the application component in response to the indication from the application component ... without action by the client," the Office has failed to establish obviousness.

For at least this reason claim 21 is allowable.

CONCLUSION

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By _____
Gregory L. Maurer
Registration No. 43,781

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446

(80683.1USORG)

Marked-up Version of Amended Claims

Pursuant to 37 C.F.R. §1.121

8. (Once Amended) [The computer operating environment of claim 5 further comprising:] In a computer, a computer operating environment for scalable, component-based server applications, comprising:

a run-time service for executing an application component in a process, the application component having a state and implementing a set of functions;

an instance creation service operative, responsive to a request of a client, to return a reference to the application component through the run-time service to the client, whereby the client calls functions of the application component indirectly through the run-time service using the reference to initiate work by the application component;

the run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client; and

a component context associated by the run-time service with the application component and providing an interface having a member function that the application component calls to initiate the indication.

15. (Once Amended) [The method of claim 13 wherein the step of receiving the indication comprises:] In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:

encapsulating function code and a processing state for the work in a component;

providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;

receiving an indication from the component that the work by the component is complete; and

discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete;

wherein the step of receiving the indication includes providing a context object containing data representing a context of the component and having an integration interface for receiving a call of the component to indicate that the work by the component is complete.

18. (Once Amended) In a computer, a system service for providing an execution environment for scalable application components, comprising:

code responsive to a request from a client program to create an application component for returning to the client program a reference through the system service to the application component;

code responsive to a call from the client program using the reference for initiating processing of work by the application component, the application component producing a processing state during processing the work;

code for receiving an indication from the application component that processing by the application component of the work is complete; and

code for destroying the processing state of the application program in response to the indication from the application component without action from the client program.

25. (New) In a computer having a main memory, an application component having a state and function code for performing work responsive to method invocations from a client, an improved method of enhancing scalability of server applications, comprising:

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without an indication from the client allowing the state of the application component to be destroyed.

26. (New) The method of claim 25, further comprising:

subsequent to the destroying step, restoring the state of the application component in main memory upon receiving a method invocation from the client.

27. (New) The method of claim 26, further comprising:

subsequent to the restoring step, destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without an indication from the client allowing the state of the application component to be destroyed.

28. (New) The method of claim 25, wherein the destroying step does not include notifying the client that the state of the application component is destroyed.